Abstract

Knowledge-based approaches to document categorization make use of well elaborated and powerful pattern languages for manual writing of classification rules. Although such classification patterns have proven useful in many practical applications, algorithms for learning classifiers from examples mostly rely on much simpler representations of classification knowledge. In this paper, we describe a learning algorithm which employs a pattern language similar to languages used for manual rule editing. We focus on the learning of three specific constructs of this pattern language, namely phrases, tolerance matches of words and substring matches of words.

Introduction

Manually writing document categorization rules is labor intensive and requires much expertise. This caused a growing research interest in learning systems for document categorization. Most of these systems transform pre-classified example documents into a propositional attribute-value representation. Simple attributes indicate whether (or, how often) a certain word occurs in a given document. This representation is equivalent to representing documents by the set (or, multi-set) of words they contain. Based on this representation, propositional learning techniques construct a classifier.

Recently, some authors proposed to extend the conventional representation by more complex document features such as multi-word terms or phrases (Finch 1995; Cohen 1996; Cohen & Singer 1996; Riloff 1996). Basically, there are two ways for integrating such features into the learning process. First, they can be computed in a preprocessing step, and thus be represented extensionally. Preprocessing could, e.g., add a fixed number of new attributes to the document representation, each attribute corresponding to an important phrase. For computational reasons, only a relatively small number of phrases can be selected. Since the selection is done prior to learning, it is not possible to take into account information revealed within the learning process. The second way seems much more promising: The document representation is enriched by word positions, and the learning algorithm is enabled to compose phrases context-sensitively, as they are needed during learning. This way, the algorithm has to its disposal the whole expressiveness of the underlying pattern language.

In order to implement these ideas, we have developed a prototypical rule learner relying on a document representation which preserves the order of words. Our learning algorithm is provided with specific hypothesis refinement operators which derive important document properties context-sensitively as needed (as a human would do it). One of these operators allows to build phrases, many other operators also seem useful. In our system, this is reflected by an extensible, modular system architecture which provides a simple interface for integrating additional operators.

In contrast to other approaches, our system PET (Pattern Extraction for Texts) allows to learn classification patterns as they are used for hand-crafted categorization rules (cf. (Hayes & Weinstein 1991), (Agne & Hein 1998)). Essentially, the relational learning paradigm as realized in Inductive Logic Programming (ILP) (cf. (Nienhuys-Cheng & de Wolf 1997)) provides an equivalent expressiveness for formulation of classification knowledge. However, our pattern language offers the advantage of rather compact notions with an implicit language bias towards document categorization problems. It is easy to read and understand which simplifies explanation of classification results and manual editing of classifiers. Thus, it allows future interactive scenarios in which the user and the learning system work together when designing categorization rules.

Our learning process is based on the successive refinement of hypotheses applying a number of specialization and generalization operators. In (Junker & Abecker 1997), we described our experiments with different generalization operators that exploit background knowledge provided by the electronic thesaurus WordNet. This allows, e.g., to generalize from the occurrence of the word “apple” to the occurrence of any fruit in a document.

In this paper, we present three operators for the construction of classification patterns which have proven to be useful in manual pattern creation: one operator for
Building phrases, one operator for tolerance matches of words, and one operator for substring matches of words. While forming phrases is of general interest, tolerance matches are useful for the classification of text documents with OCR errors; learning rules which test on the occurrence of word parts is of particular interest when dealing with word composites as they are common, e.g., in German texts.

Patterns for Document Categorization
In the following section, we will describe the basics of our pattern language for document categorization. Only the constructs to be learned will be introduced. A description of the full language can be found in (Wenzel & Junker 1997).

In contrast to the traditional approaches we represent a document $t$ by a sequence of words $t = (t_1, \ldots, t_n)$ for learning. A document only containing the text “this is a document” is represented by $t = (\text{this}, \text{is}, \text{a document}).$ The representation subsumes the widely used attribute-value representation of documents, but in addition preserves the order of words.

Syntax of Patterns
The syntax of patterns $P$ is defined as follows ($W$ denotes the set of natural language words):

- $w \in P$ if $w \in W$
- $p_1 \land p_2 \in P$ if $p_1, p_2 \in P$ (conjunction operator)
- $w : \text{tol} n \in P$ if $w \in W$ and $n \in \mathbb{N}_0$ (tolerance operator)
- $w : \text{sub} \in P$ if $w \in W$ (substring operator)
- $p_1 : \text{disth} n p_2 \in P$ if $p_1, p_2 \in P$ and $n \in \mathbb{N}_0$ (phrase operator)$^1$

Matching of Patterns
The matching of a pattern at position $i$ in a text $t = (t_1, \ldots, t_n)$ is defined as follows:

- $w$ matches $t$ at position $i$ if $w = t_i$
- $w : \text{tol} n$ matches $t$ at position $i$ if $ld(t_i, w) \leq n$
- $w : \text{sub}$ matches $t$ at position $i$ if $w$ is substring of $t_i$

The Levenshtein distance of two words $ld(w_1, w_2)$ as used in the definition of substring matches defines the distance between two words based on the minimum number of edit operations needed to transform the words into each other. Valid edit operations are insertion, deletion and substitution of a character. For instance, the words “INVOICE” and “1NVOIC” have a Levenshtein distance of 2. There is an efficient scheme to compute the Levenshtein distance with the help of a matrix (Sankoff & Kruskal 1983).

If a pattern matches at a position $i$ in $t$ we also say that it matches $t$ from position $i$ to position $j$. This formulation simplifies the definition of matching for the phrase operator:

- $p_1 : \text{disth} n p_2$ matches $t$ from position $i$ to position $j$, if $p_1$ matches $t$ from position $i$ to position $k$ and $p_2$ matches $t$ from position $k'$ to position $j$ with $k < k' \leq k + n + 1$

In order to simplify the description of our learning algorithm we also introduce the pattern true, which matches in every document.

Now, we can describe a category $c$ by a set of patterns. Whenever at least one pattern of the set matches in a document, the document belongs to $c$. A set of patterns $\{p_1, \ldots, p_n\}$ for a category $c$ can also be written as a set of categorization rules $\{c \leftarrow p_1, \ldots, c \leftarrow p_n\}$.

Symbolic Rule Learning for Document Categorization
Figure 1 illustrates a generic symbolic rule learner for document categorization. Propositional and relational rule learners for document categorization can both be seen as instances of this model. The input of the rule induction system is a set of pre-classified sample documents represented in an own representation language. In addition to the sample documents, background knowledge can also be formulated in an own representation language. The output of the rule learner is a set of categorization rules. The core of the system consists of a module which generates pattern hypotheses (or rule hypotheses, respectively) formulated in the hypothesis language $H$. This module is starting with a set of hypotheses. By successively applying refinements to hypotheses, new hypotheses are constructed. Refinements are functions of the form $R : H \rightarrow 2^H$, which assign to each hypothesis a set of new hypotheses. These hypotheses can be specializations as well as generalizations of the input hypothesis. The conjunction of a hypothesis with another hypothesis is an example for a simple refinement operator. The search is determined by the search algorithm (e.g., hill climbing or beam search), the search strategy (when to apply which operators), and the search heuristics (e.g., the accuracy on the training samples).

Our rule induction algorithm is an instance of the generic rule learner for document categorization. As described earlier, documents are represented by word sequences. The hypothesis space of our algorithm consists of pattern sets. For hypothesis construction, we

$^1$The identifier :disth is an abbreviation of horizontal distance. The full pattern language also supports vertical word distances.
use the separate-and-conquer strategy: First, the hypothesis true is refined to a 'good' hypothesis by applying refinement operators (conquer step). This is done using the beam search algorithm with beam size $b$. Our search heuristic consists of two parts: The primary evaluation criterion for a hypothesis is its accuracy on the training set. In addition, patterns are only accepted if they fulfill the significance criterion (we use the likelihood measure for testing on significance, cf. (Clark & Niblett 1989)). When a pattern hypothesis cannot be refined anymore to a better and significant hypothesis, this pattern is used to build a new categorization rule. Next, all positive examples covered by this rule are removed from the training set and the algorithm iterates on the remaining examples (separate step). This is repeated until no new pattern hypothesis can be found anymore.

The basic refinement function is $S_{\wedge,W}$:

$$S_{\wedge,W} : H \rightarrow 2^H, \ h \mapsto \{h \land r | r \in W\}$$

$W$ is a subset of $W$ and denotes the strongest 150 words indicating the target class in the actual conquer step. For word selection we used the $m$-estimate with $m = 10$ (Förnkranz 1996). The number 150 as well as the measure used for word selection were chosen after some initial experiments and not subject to further investigations. It is important to note that the words in $W$ are not chosen once in a preprocessing step, since a new set $W$ is determined in each conquer step.

Figure 2 shows our algorithm in more detail. The main difference to standard propositional rule learners is that we do not only refine by conjunction. After each refinement of a pattern with a word $w$ by conjunction, additional refinements are applied to $w$ in order to find useful pattern replacements of $w$. Such replacements can be more general as well as more specific than $w$ itself. We have defined refinements for learning phrases, tolerance matches, and substring matches. The learning of these three types of patterns is discussed in the next three sections.

## Learning Phrases

In traditional approaches to learning document categorizers, documents are represented by word sets. This means that a categorizer cannot rely on any word order and word distance information. Since we represent documents by word sequences, we can use this information to test on the occurrence of specific phrases. Phrases in our sense are expressions that can be built using the $disth$ operator of our pattern language.

Phrases are built recursively from single words. First, these are extended to “two-word phrases” by adding words to the left and to the right within different maximum distances. An overall maximum distance for the tests is given by the parameter $m$. The “two-word phrases” then are extended again on both sides. This iterates up to a maximum phrase length. The following refinement function refine, computes all phrase refinements of a word $h$ up to the length $i$.

$$\text{refine}_1(h) = \{h\}$$
$$\text{refine}_i(h) = \{h' : \text{disth} j w | h' \in \text{refine}_{i-1}(h), w \in W, j \leq m\}$$
$$\cup \{w : \text{disth} j h' | h' \in \text{refine}_{i-1}(h), w \in W, j \leq m\}$$

For our experiments, we used the Reuters corpus, a collection of Reuters newswire articles the use of which has a long tradition in text classification. Since the end of 1996, a revised version of this corpus is available, called Reuters-21578$^2$. We used the “ModApte” splitting proposed in the documentation, separating the corpus into 9,603 training and 3,299 test documents. For testing our classification algorithm, we relied on the TOPICS set of classes. The beam size $b$ was set to

learnRuleSet(Examples, targetClass)
let RuleSet=\{
repeat let bestPattern = findBestPattern(Examples, targetClass)
if bestPattern is significant then let RuleSet = RuleSet \cup targetClass \leftarrow bestPattern
let Examples = Examples - (all documents in Examples for which targetClass \leftarrow bestPattern is true)
until bestPattern is not significant

findBestPattern(Examples, targetClass)
let PatternHypotheses = \{true\}
let \(W\) = the 150 strongest words according to \(m\)-estimate indicating targetClass in Examples
repeat let PatternHypothesesOld = PatternHypotheses
let NewPatternHypotheses = \bigcup_{p \in \text{PatternHypotheses}} S_{w,p}(h)
let RefinedNewPatternHypotheses = \{
foreach patternHypothesis h \& w in NewPatternHypotheses
let RefinedNewPatternHypotheses = RefinedNewPatternHypotheses \cup \{h \& p | p \in \text{refine}(w)\}
let PatternHypotheses = PatternHypotheses \cup NewPatternHypotheses \cup RefinedNewPatternHypotheses
let PatternHypotheses = best \(b\) significant pattern hypotheses in PatternHypotheses
until PatternHypotheses = PatternHypothesesOld
return (best pattern hypothesis in PatternHypotheses)

Figure 2: A simple rule induction algorithm for document categorization with an interface to different word refinement operators

50. Table 1 compares the results we obtained in the 8 most frequent classes and the overall micro-averaged results on the whole test set. The table shows the results for the original setting without phrase learning capabilities and the results when learning phrases with \(m = 0\) and \(m = 3\). The columns indicate recall \((\text{rec})\), precision \((\text{prec})\) and \(f_\beta\) with \(\beta = 1\) for each experiment (for details on the measures used see, e.g., (Lewis 1992)). The maximum phrase length was set to \(i = 3\) for all experiments.

Learning Tolerance Matches
In office automation, one is faced with paper documents which have to be categorized using OCR output (Dengel et al. 1995). Here, the problem arises that words often contain character recognition errors. The Levenshtein distance is very useful for handling such errors. It allows to test with some tolerance whether a word occurs in a document. However, with a too small tolerance words often cannot be caught. On the other hand, a too big tolerance increases the risk of confusing a word with a different word in the domain. It is clear that the optimal tolerance of a word strongly depends on the concrete word and the domain. Thus, it is desirable to assign a specific maximum tolerance to each word.

For the learning of useful tolerance matches for a categorization task we can rely on word tests which are already good discriminators by themselves. The refinement of words by the tolerance operator is defined as follows:

\[
\text{refine}(w) = \{ w : \text{tol } i \mid i \leq \text{length of } w \text{ in characters} \}
\]

For the experimental evaluation of this refinement operator we relied on a corpus of 1741 German business letters each falling into exactly one of 9 categories: order confirmations (352 documents), offers (186), delivery notes (380), invoices (714), and 5 rather small categories. The letters were scanned using the commercial OCR software ScanWorX and arbitrarily split into a training set of 1160 and a test set of 581 documents. Since determining the Levenshtein distance is computationally expensive, the beam size \(b\) was reduced to 5 for these experiments. Table 2 shows the results on the test set with and without tolerance operators in comparison.

Learning Substring Matches
In German language, especially in technical domains, composites are very common. Composites are words which are basically built by just concatenating multiple words to one single word, e.g., the German words “Text” and “Kategorisierung” are used to build the word “Textkategorisierung” (text categorization). When categorizing German documents, we are not only interested in full words, but also in composite parts which are good discriminators. With the pattern \(w:sub\) we are able to find out those documents in which \(w\) occurs as a substring. Although we do not test whether \(w\) is a valid word part of a valid composite, in practice this is a good approximation.

In learning patterns, the problem arises to find useful substring tests. For the refinement of words to substring matches, we rely on two heuristics:

- if a word \(w\) is a good discriminator, there is an evi-


dence that $w$ is a discriminating composite part;

- If a word $w$ is a good discriminator, there is an evidence that some substring of it is a discriminating composite part. For efficiency, we only test those substrings of $w$ which also occur as single words in the training examples. In experiments it turned out that this is not a serious restriction, since for most composites in a reasonably sized training set all composite parts also occur in isolation.

Using these heuristics, the refinement operator for the introduction of substring matches can be written as follows ($\mathcal{W}$ denotes the set of words occurring in the training set):

$$\text{refine}(w) = \{w : \text{sub}\} \cup \{w' : \text{sub} | w' \in \mathcal{W} \text{ and } w' \text{ is substring of } w\}$$

For the experimental evaluation, we relied on a collection of 1090 German abstracts of technical reports falling into six categories. The document length varies between 25 and 334 words, on average 118 words. The categories included between 125 and 280 sample documents. The corpus was arbitrarily divided into a training set of 506 documents and a test set of 506 documents. The beam size was set to $b = 50$. Table 3 shows the improvements achieved on the test set by introducing the refinement operator for testing on substrings.

In additional experiments, we compared substring matching with a morphological decomposition. Morphological decomposition relies on a hand-crafted morphological lexicon. In contrast to our approach, it can handle unregular flexions and does not rely on word parts occurring in isolation. Furthermore, it is more restrictive than substring matching: In general, only valid word stems of valid composites are taken into account. For the morphological decomposition we used the tool Morphic-Plus (Lutzy 1995). Since we were interested in how the morphological analysis would perform in the optimal case, the lexicon was completed for our domain. Using Morphic-Plus, all words (and in particular composites) were reduced in a pre-processing step to their respective stems. With this setting, we obtained an overall result of $f_1 = 65.2$ (62.0% recall and 68.7% precision). Note that using substring matches $f_1$ improved by 9.1% (from 52.8% to 61.9%) while with morphological decomposition we gained another 3.3%. Relating this gain to the enormous amount of manual effort to complete the morphological lexicon, substring matching is a very competitive technique.

### Evaluation of Experiments

Our experiments showed slight overall improvements by introducing the new refinement operators. However, we relied upon a quite simple core system which has only a very basic pre-pruning technique to avoid overfitting. Several authors emphasized the importance of powerful post-pruning techniques for coping with this problem ((Cohen 1996), (Fürnkranz 1997)). Just recently, (Joachims 1998) presented highly impressive results in document categorization with a statistical learner. Also in this system, appropriate dealing with the overfitting problem seems to be crucial. Our experiments confirm the importance of protection against overfitting. When looking at the produced classification rules it turns out that they tend to be very long and too much tailored

---

### Table 1: Results when refining words by phrases

<table>
<thead>
<tr>
<th>class</th>
<th>rec</th>
<th>prec</th>
<th>$f_1$</th>
<th>rec</th>
<th>prec</th>
<th>$f_1$</th>
<th>rec</th>
<th>prec</th>
<th>$f_1$</th>
<th>rec</th>
<th>prec</th>
<th>$f_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>earn</td>
<td>97.1</td>
<td>93.6</td>
<td>95.3</td>
<td>97.5</td>
<td>94.1</td>
<td>95.8</td>
<td>98.0</td>
<td>94.3</td>
<td>96.1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>acq</td>
<td>81.1</td>
<td>85.4</td>
<td>83.2</td>
<td>82.6</td>
<td>86.3</td>
<td>84.4</td>
<td>83.3</td>
<td>88.2</td>
<td>85.7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>crude</td>
<td>74.1</td>
<td>85.4</td>
<td>79.3</td>
<td>73.5</td>
<td>84.2</td>
<td>78.5</td>
<td>83.1</td>
<td>80.1</td>
<td>81.6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>money fx</td>
<td>41.3</td>
<td>67.9</td>
<td>51.4</td>
<td>50.8</td>
<td>69.5</td>
<td>50.8</td>
<td>50.3</td>
<td>69.2</td>
<td>58.3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>grain</td>
<td>91.9</td>
<td>90.7</td>
<td>91.3</td>
<td>91.9</td>
<td>90.1</td>
<td>91.0</td>
<td>94.0</td>
<td>90.3</td>
<td>92.1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>interest</td>
<td>32.1</td>
<td>75.0</td>
<td>44.9</td>
<td>33.6</td>
<td>71.0</td>
<td>45.6</td>
<td>37.4</td>
<td>74.2</td>
<td>49.7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>trade</td>
<td>52.1</td>
<td>64.9</td>
<td>57.8</td>
<td>59.0</td>
<td>67.0</td>
<td>62.7</td>
<td>57.3</td>
<td>74.4</td>
<td>64.7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>overall (micro avg.)</td>
<td>75.9</td>
<td>84.0</td>
<td>79.7</td>
<td>77.9</td>
<td>84.0</td>
<td>80.8</td>
<td>78.8</td>
<td>85.0</td>
<td>81.8</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table 2: Results when refining words by tolerance matches

<table>
<thead>
<tr>
<th>class</th>
<th>rec</th>
<th>prec</th>
<th>$f_1$</th>
<th>rec</th>
<th>prec</th>
<th>$f_1$</th>
<th>rec</th>
<th>prec</th>
<th>$f_1$</th>
<th>rec</th>
<th>prec</th>
<th>$f_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>offer</td>
<td>81.2</td>
<td>92.9</td>
<td>86.7</td>
<td>75.0</td>
<td>82.8</td>
<td>78.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>confirmation of order</td>
<td>61.5</td>
<td>72.0</td>
<td>66.4</td>
<td>70.1</td>
<td>70.1</td>
<td>70.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>delivery note</td>
<td>47.9</td>
<td>66.7</td>
<td>55.8</td>
<td>55.4</td>
<td>72.0</td>
<td>62.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>invoice</td>
<td>73.0</td>
<td>75.5</td>
<td>74.2</td>
<td>75.6</td>
<td>77.2</td>
<td>76.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>overall (micro avg.)</td>
<td>67.8</td>
<td>74.6</td>
<td>71.0</td>
<td>70.5</td>
<td>74.5</td>
<td>72.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
In addition to overspecialization, we also found a phenomenon resulting from our use of generalization operators. Consider, e.g., the bad results for the “offer” class in the experiments with tolerance matches (Table 2). They cause a significant degradation of the overall effectiveness in this experiment. The examination of the produced classification rules revealed that, here, an overgeneralization —the logical counterpart of overfitting— took place because of a too “greedy” search strategy.

### Conclusion

We presented an approach for learning pattern-based document classifiers. Compared to most conventional systems, we extend the expressiveness of both document representation and classifier language. The most distinctive feature of our rule learner is that it context-sensitively applies domain-specific refinement operators. New refinement operators can easily be plugged into the core system as modules. In contrast to other rule learners used for document categorization, the refinements can be specializations as well as generalizations. In this paper, we focused on three refinement operators, namely for phrase building, tolerance matches, and substring matches.

The evaluation of our experiments revealed that overfitting and overgeneralization are important problems to tackle. For future improvements of our approach it seems interesting to investigate how our refinement operators play together with more sophisticated search strategies and pruning techniques.

### References


